OpenShift Container Platform 4.5

Jaeger

Jaeger installation, usage, and release notes

OpenShift Container Platform 4.5 Jaeger

Jaeger installation, usage, and release notes

Legal Notice

Abstract

This document provides information on how to use Jaeger in OpenShift Container Platform.

Table of Contents

CHAPTER 1. JAEGER RELEASE NOTES

1.1. JAEGER OVERVIEW

As a service owner, you can use Jaeger to instrument your services to gather insights into your service architecture. Jaeger is an open source distributed tracing platform that you can use for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications.

Using Jaeger lets you perform the following functions:

- Monitor distributed transactions

- Optimize performance and latency

- Perform root cause analysis

Jaeger is based on the vendor-neutral OpenTracing APIs and instrumentation.

1.2. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, or with OpenShift Container Platform in general, visit the Red Hat Customer Portal . From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.

- Submit a support case to Red Hat Support.

- Access other product documentation.

If you have a suggestion for improving this documentation or have found an error, please submit a Bugzilla report against the **OpenShift Container Platform** product for the **Documentation** component. Please provide specific details, such as the section name and OpenShift Container Platform version.

1.2.1. New features OpenShift Jaeger 1.17.4

This release of OpenShift Jaeger addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.2. New features OpenShift Jaeger 1.17.3

This release of OpenShift Jaeger addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.3. New features OpenShift Jaeger 1.17.2

This release of OpenShift Jaeger addresses Common Vulnerabilities and Exposures (CVEs) and bug fixes.

1.2.4. New features OpenShift Jaeger 1.17.1

This release of OpenShift Jaeger adds support for installing Jaeger as a standalone solution, rather than as a component of Red Hat OpenShift Service Mesh.

1.3. JAEGER KNOWN ISSUES

These limitations exist in Jaeger:

- While Kafka publisher is included as part of Jaeger, it is not supported.

- Apache Spark is not supported.

- Only self-provisioned Elasticsearch instances are supported. External Elasticsearch instances are not supported in this release.

These are the known issues in Jaeger:

- TRACING-1166 It is not currently possible to use the Jaeger streaming strategy within a disconnected environment. When a Kafka cluster is being provisioned, it results in a error: **Failed to pull image registry.redhat.io/amq7/amq-streams-kafka-24-rhel7@sha256:f9ceca004f1b7dccb3b82d9a8027961f9fe4104e0ed69752c0bdd8078b4a1076**.

- TRACING-809 Jaeger Ingester is incompatible with Kafka 2.3. When there are two or more instances of the Jaeger Ingester and enough traffic it will continuously generate rebalancing messages in the logs. This is due to a regression in Kafka 2.3 that was fixed in Kafka 2.3.1. For more information, see Jaegertracing-1819.

CHAPTER 2. JAEGER ARCHITECTURE

2.1. JAEGER ARCHITECTURE

Every time a user takes an action in an application, a request is executed by the architecture that may require dozens of different services to participate in order to produce a response. Jaeger lets you perform distributed tracing, which records the path of a request through various microservices that make up an application.

Distributed tracing is a technique that is used to tie the information about different units of work together — usually executed in different processes or hosts — to understand a whole chain of events in a distributed transaction. Developers can visualize call flows in large microservice architectures with distributed tracing. It's valuable for understanding serialization, parallelism, and sources of latency.

Jaeger records the execution of individual requests across the whole stack of microservices, and presents them as traces. A *trace* is a data/execution path through the system. An end-to-end trace is comprised of one or more spans.

A *span* represents a logical unit of work in Jaeger that has an operation name, the start time of the operation, and the duration, as well as potentially tags and logs. Spans may be nested and ordered to model causal relationships.

2.1.1. Jaeger overview

As a service owner, you can use Jaeger to instrument your services to gather insights into your service architecture. Jaeger is an open source distributed tracing platform that you can use for monitoring, network profiling, and troubleshooting the interaction between components in modern, cloud-native, microservices-based applications.

Using Jaeger lets you perform the following functions:

- Monitor distributed transactions

- Optimize performance and latency

- Perform root cause analysis

Jaeger is based on the vendor-neutral OpenTracing APIs and instrumentation.

2.1.2. Jaeger features

Jaeger tracing provides the following capabilities:

- Integration with Kiali – When properly configured, you can view Jaeger data from the Kiali console.

- High scalability – The Jaeger backend is designed to have no single points of failure and to scale with the business needs.

- Distributed Context Propagation – Lets you connect data from different components together to create a complete end-to-end trace.

- Backwards compatibility with Zipkin – Jaeger has APIs that enable it to be used as a drop-in replacement for Zipkin, but Red Hat is not supporting Zipkin compatibility in this release.

2.1.3. Jaeger architecture

Jaeger is made up of several components that work together to collect, store, and display tracing data.

- **Jaeger Client** (Tracer, Reporter, instrumented application, client libraries)– Jaeger clients are language specific implementations of the OpenTracing API. They can be used to instrument applications for distributed tracing either manually or with a variety of existing open source frameworks, such as Camel (Fuse), Spring Boot (RHOAR), MicroProfile (RHOAR/Thorntail), Wildfly (EAP), and many more, that are already integrated with OpenTracing.

- **Jaeger Agent** (Server Queue, Processor Workers) – The Jaeger agent is a network daemon that listens for spans sent over User Datagram Protocol (UDP), which it batches and sends to the collector. The agent is meant to be placed on the same host as the instrumented application. This is typically accomplished by having a sidecar in container environments like Kubernetes.

- **Jaeger Collector** (Queue, Workers) – Similar to the Agent, the Collector is able to receive spans and place them in an internal queue for processing. This allows the collector to return immediately to the client/agent instead of waiting for the span to make its way to the storage.

- **Storage** (Data Store) – Collectors require a persistent storage backend. Jaeger has a pluggable mechanism for span storage. Note that for this release, the only supported storage is Elasticsearch.

- **Query** (Query Service) – Query is a service that retrieves traces from storage.

- **Ingester** (Ingester Service) – Jaeger can use Apache Kafka as a buffer between the collector and the actual backing storage (Elasticsearch). Ingester is a service that reads data from Kafka and writes to another storage backend (Elasticsearch).

- **Jaeger Console** – Jaeger provides a user interface that lets you visualize your distributed tracing data. On the Search page, you can find traces and explore details of the spans that make up an individual trace.

CHAPTER 3. JAEGER INSTALLATION

3.1. INSTALLING JAEGER

You can install Jaeger on OpenShift Container Platform in either of two ways:

- You can install Jaeger as part of Red Hat OpenShift Service Mesh. Jaeger is included by default in the Service Mesh installation. To install Jaeger as part of a service mesh, follow the Red Hat Service Mesh Installation instructions.

- If you do not want to install a service mesh, you can use the Jaeger Operator to install the Red Hat build of Jaeger by itself. To install Jaeger without a service mesh, use the following instructions.

3.1.1. Prerequisites

Before you can install OpenShift Jaeger, review the installation activities, and ensure that you meet the prerequisites:

- Possess an active OpenShift Container Platform subscription on your Red Hat account. If you do not have a subscription, contact your sales representative for more information.

- Review the OpenShift Container Platform 4.5 overview.

- Install OpenShift Container Platform 4.5.

  - Install OpenShift Container Platform 4.5 on AWS

  - Install OpenShift Container Platform 4.5 on user-provisioned AWS

  - Install OpenShift Container Platform 4.5 on bare metal

  - Install OpenShift Container Platform 4.5 on vSphere

- Install the version of the OpenShift Container Platform command line utility (the **oc** client tool) that matches your OpenShift Container Platform version and add it to your path.

- An account with the **cluster-admin** role.

3.1.2. Jaeger installation overview

The steps for installing OpenShift Jaeger are as follows:

- Review the documentation and determine your deployment strategy.

- If your deployment strategy requires persistent storage, install the Elasticsearch Operator via the OperatorHub.

- Install the Jaeger Operator via the OperatorHub.

- Modify the Jaeger YAML file to support your deployment strategy.

- Deploy one or more instances of Jaeger to your OpenShift Container Platform environment.

3.1.3. Installing the Elasticsearch Operator

The default Jaeger deployment uses in-memory storage because it is designed to be installed quickly for those evaluating Jaeger, giving demonstrations, or using Jaeger in a test environment. If you plan to use Jaeger in production, you must install a persistent storage option, in this case, Elasticsearch.

Prerequisites

- Access to the OpenShift Container Platform web console.

- An account with the **cluster-admin** role.

> **WARNING**
>
> Do not install Community versions of the Operators. Community Operators are not supported.

> **NOTE**
>
> If you have already installed the Elasticsearch Operator as part of OpenShift cluster logging, you do not need to install the Elasticsearch Operator again. The Jaeger Operator will create the Elasticsearch instance using the installed Elasticsearch Operator.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.

2. Navigate to **Operators → OperatorHub**.

3. Type **Elasticsearch** into the filter box to locate the Elasticsearch Operator.

4. Click the **Elasticsearch Operator** provided by Red Hat to display information about the Operator.

5. Click **Install**.

6. On the **Install Operator** page, select the **A specific namespace on the cluster** option and then select **openshift-operators-redhat** from the menu.

7. Select the **Update Channel** that matches your OpenShift Container Platform installation. For example, if you are installing on OpenShift Container Platform version 4.5, select the 4.5 update channel.

8. Select the **Automatic** Approval Strategy.

> **NOTE**
>
> The Manual approval strategy requires a user with appropriate credentials to approve the Operator install and subscription process.

9. Click **Install**.

10. On the **Installed Operators** page, select the **openshift-operators-redhat** project. Wait until you see that the Elasticsearch Operator shows a status of "InstallSucceeded" before continuing.

3.1.4. Installing the Jaeger Operator

To install Jaeger you use the OperatorHub to install the Jaeger Operator.

By default the Operator is installed in the **openshift-operators** project.

Prerequisites

- Access to the OpenShift Container Platform web console.

- An account with the **cluster-admin** role.

- If you require persistent storage, you must also install the Elasticsearch Operator before installing the Jaeger Operator.

> ⚠ **WARNING**
>
> Do not install Community versions of the Operators. Community Operators are not supported.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.

2. Navigate to **Operators → OperatorHub**.

3. Type **Jaeger** into the filter to locate the Jaeger Operator.

4. Click the **Jaeger Operator** provided by Red Hat to display information about the Operator.

5. Click **Install**.

6. On the **Install Operator** page, select **All namespaces on the cluster (default)** This installs the Operator in the default **openshift-operators** project and makes the Operator available to all projects in the cluster.

7. Select the **stable** Update Channel. This will automatically update Jaeger as new versions are released. If you select a maintenance channel, for example, **1.17-stable**, you will receive bug fixes and security patches for the length of the support cycle for that version.

   - Select an Approval Strategy. You can select **Automatic** or **Manual** updates. If you choose Automatic updates for an installed Operator, when a new version of that Operator is available, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select Manual updates, when a newer version of an Operator is available, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

8. Click **Install**.

9. On the **Subscription Overview** page, select the **openshift-operators** project. Wait until you see that the Jaeger Operator shows a status of "InstallSucceeded" before continuing.

3.2. CONFIGURING AND DEPLOYING JAEGER

The Jaeger Operator includes a custom resource definition (CRD) file that defines the architecture and configuration settings for the Jaeger resources. You can either install the default configuration or modify the file to better suit your business requirements.

Jaeger has predefined deployment strategies. You specify a deployment strategy in the custom resource file. When you create a Jaeger instance the Operator uses this configuration file to create the objects necessary for the deployment.

Jaeger custom resource file showing deployment strategy

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production  1
```

1 The Jaeger Operator currently supports the following deployment strategies:

- **allInOne** (Default) – This strategy is intended for development, testing, and demo purposes. The main backend components, Agent, Collector and Query service, are all packaged into a single executable which is configured (by default) to use in-memory storage.

  > **NOTE**
  >
  > In-memory storage is not persistent, which means that if the Jaeger instance shuts down, restarts, or is replaced, that your trace data will be lost. And in-memory storage cannot be scaled, since each pod has its own memory. For persistent storage, you must use the **production** or **streaming** strategies, which use Elasticsearch as the default storage.

- **production** – The production strategy is intended for production environments, where long term storage of trace data is important, as well as a more scalable and highly available architecture is required. Each of the backend components is therefore deployed separately. The Agent can be injected as a sidecar on the instrumented application or as a daemonset. The Query and Collector services are configured with a supported storage type - currently Elasticsearch. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

- **streaming** – The streaming strategy is designed to augment the production strategy by providing a streaming capability that effectively sits between the Collector and the backend storage (Elasticsearch). This provides the benefit of reducing the pressure on the backend storage, under high load situations, and enables other trace post-processing capabilities to tap into the real time span data directly from the streaming platform (AMQ Streams/ Kafka).

> **NOTE**
>
> There are two ways to install Jaeger, as part of a service mesh or as a stand alone component. If you have installed Jaeger as part of Red Hat OpenShift Service Mesh, you must configure and deploy Jaeger as part of the ServiceMeshControlPlane.

3.2.1. Deploying the default Jaeger strategy from the web console

The custom resource definition (CRD) defines the configuration used when you deploy an instance of Jaeger. The default CR for Jaeger is named **jaeger-all-in-one-inmemory** and it is configured with minimal resources to ensure that you can successfully install it on a default OpenShift Container Platform installation. You can use this default configuration to create a Jaeger instance that uses the **AllInOne** deployment strategy, or you can define your own custom resource file.

> **NOTE**
>
> In-memory storage is not persistent, which means that if the Jaeger pod shuts down, restarts, or is replaced, that your trace data will be lost. For persistent storage, you must use the **production** or **streaming** strategies, which use Elasticsearch as the default storage.

Prerequisites

- The Jaeger Operator must be installed.

- Review the instructions for how to customize the Jaeger installation.

- An account with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.

2. Create a new project, for example **jaeger-system**.

   a. Navigate to **Home → Projects**.

   b. Click **Create Project**.

   c. Enter **jaeger-system** in the **Name** field.

   d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.

4. If necessary, select **jaeger-system** from the Project menu. You may have to wait a few moments for the Operators to be copied to the new project.

5. Click the OpenShift Jaeger Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.

6. Under **Jaeger** click **Create Instance**.

7. On the **Create Jaeger** page, to install using the defaults, click **Create** to create the Jaeger instance.

8. On the **Jaegers** page, click the name of the Jaeger instance, for example, **jaeger-all-in-one-inmemory**.

9. On the **Jaeger Details** page, click the **Resources** tab. Wait until the Pod has a status of "Running" before continuing.

3.2.1.1. Deploying default Jaeger from the CLI

Follow this procedure to create an instance of Jaeger from the command line.

Prerequisites

- An installed, verified OpenShift Jaeger Operator.

- Access to the OpenShift CLI (**oc**) that matches your OpenShift Container Platform version.

- An account with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

   ```
   $ oc login https://{HOSTNAME}:8443
   ```

2. Create a new project named **jaeger-system**.

   ```
   $ oc new-project jaeger-system
   ```

3. Create a custom resource file named **jaeger.yaml** that contains the following text:

   Example jaeger-all-in-one.yaml

   ```
   apiVersion: jaegertracing.io/v1
   kind: Jaeger
   metadata:
     name: jaeger-all-in-one-inmemory
   ```

4. Run the following command to deploy Jaeger:

   ```
   $ oc create -n jaeger-system -f jaeger.yaml
   ```

5. Run the following command to watch the progress of the Pods during the installation process:

   ```
   $ oc get pods -n jaeger-system -w
   ```

   Once the installation process has completed, you should see output similar to the following:

   ```
   NAME                                    READY  STATUS   RESTARTS  AGE
   jaeger-all-in-one-inmemory-cdff7897b-qhfdx  2/2    Running  0         24s
   ```

CHAPTER 4. DEPLOYING THE JAEGER PRODUCTION STRATEGY FROM THE WEB CONSOLE

The **production** deployment strategy is intended for production environments, where long term storage of trace data is important, as well as a more scalable and highly available architecture is required.

Prerequisites

- The Elasticsearch Operator must be installed.

- The Jaeger Operator must be installed.

- Review the instructions for how to customize the Jaeger installation.

- An account with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.

2. Create a new project, for example **jaeger-system**.

   a. Navigate to **Home → Projects**.

   b. Click **Create Project**.

   c. Enter **jaeger-system** in the **Name** field.

   d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.

4. If necessary, select **jaeger-system** from the Project menu. You may have to wait a few moments for the Operators to be copied to the new project.

5. Click the Jaeger Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.

6. Under **Jaeger** click **Create Instance**.

7. On the **Create Jaeger** page, replace the default **all-in-one** yaml text with your production YAML configuration, for example:

   Example jaeger-production.yaml file with Elasticsearch

   ```
   apiVersion: jaegertracing.io/v1
   kind: Jaeger
   metadata:
     name: jaeger-production
     namespace:
   spec:
     strategy: production
     ingress:
       security: oauth-proxy
     storage:
       type: elasticsearch
   ```

```
    elasticsearch:
      nodeCount: 3
      redundancyPolicy: SingleRedundancy
    esIndexCleaner:
      enabled: true
      numberOfDays: 7
      schedule: 55 23 * * *
    esRollover:
      schedule: '*/30 * * * *'
```

8. Click **Create** to create the Jaeger instance.

9. On the **Jaegers** page, click the name of the Jaeger instance, for example, **jaeger-prod-elasticsearch**.

10. On the **Jaeger Details** page, click the **Resources** tab. Wait until all the pods have a status of "Running" before continuing.

4.1. DEPLOYING JAEGER PRODUCTION FROM THE CLI

Follow this procedure to create an instance of Jaeger from the command line.

Prerequisites

- An installed, verified OpenShift Jaeger Operator.

- Access to the OpenShift CLI (**oc**).

- An account with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

   ```
   $ oc login https://{HOSTNAME}:8443
   ```

2. Create a new project named **jaeger-system**.

   ```
   $ oc new-project jaeger-system
   ```

3. Create a custom resource file named **jaeger-production.yaml** that contains the text of the example file in the previous procedure.

4. Run the following command to deploy Jaeger:

   ```
   $ oc create -n jaeger-system -f jaeger-production.yaml
   ```

5. Run the following command to watch the progress of the Pods during the installation process:

   ```
   $ oc get pods -n jaeger-system -w
   ```

   Once the installation process has completed, you should see output similar to the following:

```
NAME                                                      READY  STATUS   RESTARTS  AGE
elasticsearch-cdm-jaegersystemjaegerproduction-1-6676cf568gwhlw  2/2    Running  0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-2-bcd4c8bf5l6g6w  2/2    Running  0
10m
elasticsearch-cdm-jaegersystemjaegerproduction-3-844d6d9694hhst  2/2    Running  0
10m
jaeger-production-collector-94cd847d-jwjlj                       1/1    Running  3        8m32s
jaeger-production-query-5cbfbd499d-tv8zf                         3/3    Running  3        8m32s
```

CHAPTER 5. DEPLOYING THE JAEGER STREAMING STRATEGY FROM THE WEB CONSOLE

The **streaming** deployment strategy is intended for production environments, where long term storage of trace data is important, as well as a more scalable and highly available architecture is required.

The **streaming** strategy provides a streaming capability that sits between the collector and the storage (Elasticsearch). This reduces the pressure on the storage under high load situations, and enables other trace post-processing capabilities to tap into the real time span data directly from the streaming platform (Kafka).

Prerequisites

- The AMQ Streams Operator must be installed. If using version 1.4.0 or higher you can use self-provisioning. If otherwise, you need to create the Kafka instance.

- The Jaeger Operator must be installed.

- Review the instructions for how to customize the Jaeger installation.

- An account with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform web console as a user with the **cluster-admin** role.

2. Create a new project, for example **jaeger-system**.

   a. Navigate to **Home → Projects**.

   b. Click **Create Project**.

   c. Enter **jaeger-system** in the Name field.

   d. Click **Create**.

3. Navigate to **Operators → Installed Operators**.

4. If necessary, select **jaeger-system** from the Project menu. You may have to wait a few moments for the Operators to be copied to the new project.

5. Click the Jaeger Operator. On the **Overview** tab, under **Provided APIs**, the Operator provides a single link.

6. Under **Jaeger** click **Create Instance**.

7. On the **Create Jaeger** page, replace the default **all-in-one** yaml text with your streaming YAML configuration, for example:

Example jaeger-streaming.yaml file

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-streaming
spec:
```

```
strategy: streaming
collector:
  options:
    kafka:
      producer:
        topic: jaeger-spans
        #Note: If brokers are not defined,AMQStreams 1.4.0+ will self-provision Kafka.
        brokers: my-cluster-kafka-brokers.kafka:9092
storage:
  type: elasticsearch
ingester:
  options:
    kafka:
      consumer:
        topic: jaeger-spans
        brokers: my-cluster-kafka-brokers.kafka:9092
```

1. Click **Create** to create the Jaeger instance.

2. On the **Jaegers** page, click the name of the Jaeger instance, for example, **jaeger-streaming**.

3. On the **Jaeger Details** page, click the **Resources** tab. Wait until all the pods have a status of "Running" before continuing.

5.1. DEPLOYING JAEGER STREAMING FROM THE CLI

Follow this procedure to create an instance of Jaeger from the command line.

Prerequisites

- An installed, verified OpenShift Jaeger Operator.

- Access to the OpenShift CLI (**oc**).

- An account with the **cluster-admin** role.

Procedure

1. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

   ```
   $ oc login https://{HOSTNAME}:8443
   ```

2. Create a new project named **jaeger-system**.

   ```
   $ oc new-project jaeger-system
   ```

3. Create a custom resource file named **jaeger-streaming.yaml** that contains the text of the example file in the previous procedure.

4. Run the following command to deploy Jaeger:

   ```
   $ oc create -n jaeger-system -f jaeger-streaming.yaml
   ```

5. Run the following command to watch the progress of the Pods during the installation process:

```
$ oc get pods -n jaeger-system -w
```

Once the installation process has completed, you should see output similar to the following:

```
NAME                                                       READY  STATUS   RESTARTS  AGE
elasticsearch-cdm-jaegersystemjaegerstreaming-1-697b66d6fcztcnn  2/2    Running  0
5m40s
elasticsearch-cdm-jaegersystemjaegerstreaming-2-5f4b95c78b9gckz  2/2    Running  0
5m37s
elasticsearch-cdm-jaegersystemjaegerstreaming-3-7b6d964576nnz97  2/2    Running  0
5m5s
jaeger-streaming-collector-6f6db7f99f-rtcfm                1/1    Running  0      80s
jaeger-streaming-entity-operator-6b6d67cc99-4lm9q          3/3    Running  2
2m18s
jaeger-streaming-ingester-7d479847f8-5h8kc                 1/1    Running  0      80s
jaeger-streaming-kafka-0                                   2/2    Running  0      3m1s
jaeger-streaming-query-65bf5bb854-ncnc7                    3/3    Running  0      80s
jaeger-streaming-zookeeper-0                               2/2    Running  0      3m39s
```

5.1.1. Customizing Jaeger deployment

5.1.1.1. Jaeger default configuration options

The Jaeger custom resource (CR) defines the architecture and settings to be used when creating the Jaeger resources. You can modify these parameters to customize your Jaeger implementation to your business needs.

Jaeger generic YAML example

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: name
spec:
  strategy: <deployment_strategy>
  allInOne:
    options: {}
    resources: {}
  agent:
    options: {}
    resources: {}
  collector:
    options: {}
    resources: {}
  sampling:
    options: {}
  storage:
    type:
    options: {}
  query:
    options: {}
    resources: {}
  ingester:
```

```
    options: {}
    resources: {}
  options: {}
```

Table 5.1. Jaeger parameters

Parameter	Description	Values	Default value
apiVersion:	Version of the Application Program Interface to use when creating the object.	**jaegertracing.io/v1**	**jaegertracing.io/v1**
kind:	Defines the kind of Kubernetes object to create.	**jaeger**	
metadata:	Data that helps uniquely identify the object, including a **name** string, **UID**, and optional **namespace**.		OpenShift automatically generates the **UID** and completes the **namespace** with the name of the project where the object is created.
name:	Name for the object.	The name of your Jaeger instance.	**jaeger-all-in-one-inmemory**
spec:	Specification for the object to be created.	Contains all of the configuration parameters for your Jaeger instance. When a common definition (for all Jaeger components) is required, it is defined under the spec node. When the definition relates to an individual component, it is placed under the spec/<component> node.	N/A
strategy:	Jaeger deployment strategy	**allInOne**, **production**, or **streaming**	**allInOne**

Parameter	Description	Values	Default value
allInOne:	Because the allInOne image deploys the agent, collector, query, ingester, Jaeger UI in a single pod, configuration for this deployment should nest component configuration under the allInOne parameter.		
agent:	Configuration options that define the Jaeger agent.		
collector:	Configuration options that define the Jaeger Collector.		
sampling:	Configuration options that define the sampling strategies for tracing.		
storage:	Configuration options that define the storage. All storage related options should be placed under **storage**, rather than under the **allInOne** or other component options.		
query:	Configuration options that define the Query service.		
ingester:	Configuration options that define the Ingester service.		

The following example YAML is the minimum required to create a Jaeger instance using the default settings.

Example minimum required jaeger-all-in-one.yaml

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-all-in-one-inmemory
```

5.1.1.2. Jaeger Collector configuration options

The Jaeger Collector is the component responsible for receiving the spans that were captured by the tracer and writing them to a persistent storage (Elasticsearch) when using the **production** strategy, or to AMQ Streams when using the **streaming** strategy.

The collectors are stateless and thus many instances of Jaeger Collector can be run in parallel. Collectors require almost no configuration, except for the location of the Elasticsearch cluster.

Table 5.2. Jaeger Collector parameters

Parameter	Description	Values

spec:
  collector:
   options: {}
	Configuration options that define the Jaeger Collector.	

autoscale:	This parameter controls whether to enable/disable autoscaling for the Collector. Set to **false** to explicitly disable autoscaling.	**true**/**false**

kafka:
  producer:
   topic: jaeger-spans
	The **topic** parameter identifies the Kafka configuration used by the collector to produce the messages, and the ingester to consume the messages.	Label for the producer

kafka:
  producer:
   brokers: my-cluster-kafka-brokers.kafka:9092
	Identifies the Kafka configuration used by the Collector to produce the messages. If brokers are not specified, and you have AMQ Streams 1.4.0+ installed, Jaeger will self-provision Kafka.	

log-level:	Logging level for the collector.	**trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**
maxReplicas:	Specifies the maximum number of replicas to create when autoscaling the Collector.	Integer, for example, **100**
num-workers:	The number of workers pulling from the queue.	Integer, for example, **50**
queue-size:	The size of the Collector queue.	Integer, for example, **2000**

Parameter	Description	Values
replicas:	Specifies the number of Collector replicas to create.	Integer, for example, **5**

5.1.1.2.1. Configuring the Collector for autoscaling

You can configure the Collector to autoscale; the Collector will scale up or down based on the CPU and/or memory consumption. Configuring the Collector to autoscale can help you ensure your Jaeger environment scales up during times of increased load, and scales down when less resources are needed, saving on costs. You configure autoscaling by setting the **autoscale** parameter to **true** and specifying a value for **.spec.collector.maxReplicas** along with a reasonable value for the resources that you expect the Collector's pod to consume. If you do not set a value for **.spec.collector.maxReplicas** the Operator will set it to **100**.

By default, when there is no value provided for **.spec.collector.replicas**, the Jaeger Operator creates a Horizontal Pod Autoscaler (HPA) configuration for the Collector. For more information about HPA, refer to the Kubernetes documentation.

The following is an example autoscaling configuration, setting the Collector's limits as well as the maximum number of replicas:

Collector autoscaling example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  collector:
    maxReplicas: 5
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
```

5.1.1.3. Jaeger sampling configuration options

The Operator can be used to define sampling strategies that will be supplied to tracers that have been configured to use a remote sampler.

While all traces are generated, only a few are sampled. Sampling a trace marks the trace for further processing and storage.

> **NOTE**
>
> This is not relevant if a trace was started by the Istio proxy as the sampling decision is made there. The Jaeger sampling decision is only relevant when the trace is started by an application using the Jaeger tracer.

When a service receives a request that contains no trace context, the Jaeger tracer will start a new trace,

assign it a random trace ID, and make a sampling decision based on the currently installed sampling strategy. The sampling decision is propagated to all subsequent requests in the trace, so that other services are not making the sampling decision again.

Jaeger libraries support the following samplers:

- **Constant** – The sampler always makes the same decision for all traces. It either samples all traces (sampling.param=1) or none of them (sampling.param=0).

- **Probabilistic** – The sampler makes a random sampling decision with the probability of sampling equal to the value of the **sampling.param** property. For example, with sampling.param=0.1 approximately 1 in 10 traces will be sampled.

- **Rate Limiting** – The sampler uses a leaky bucket rate limiter to ensure that traces are sampled with a certain constant rate. For example, when sampling.param=2.0 it will sample requests with the rate of 2 traces per second.

- **Remote** – The sampler consults the Jaeger agent for the appropriate sampling strategy to use in the current service. This allows controlling the sampling strategies in the services from a central configuration in the Jaeger backend.

Table 5.3. Jaeger sampling parameters

Parameter	Description	Values	Default value

spec:
  sampling:
   options: {}
	Configuration options that define the sampling strategies for tracing.		

sampling: type:	Sampling strategy to use. (See descriptions above.)	Valid values are **const**, **probabilistic**, **ratelimiting**, and **remote**.	**remote**

sampling:
  options:
    type:
    param:
	Parameters for the selected sampling strategy. (See examples above.)	Decimal and integer values (0, .1, 1, 10)	N/A

This example defines a default sampling strategy that is probabilistic, with a 50% chance of the trace instances being sampled.

Probabilistic sampling example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: with-sampling
spec:
  strategy: allInOne
  sampling:
```

```
options:
  default_strategy:
    type: probabilistic
    param: 50
```

5.1.1.4. Jaeger storage configuration options

You configure storage for the Collector, Ingester, and Query services under **spec:storage**. Multiple instances of each of these components can be provisioned as required for performance and resilience purposes.

Restrictions

- There can be only one Jaeger with self-provisioned Elasticsearch instance per namespace.

- There can be only one Elasticsearch per namespace.

- You cannot share or reuse a OpenShift Jaeger logging Elasticsearch instance with Jaeger. The Elasticsearch cluster is meant to be dedicated for a single Jaeger instance.

NOTE

If you already have installed Elasticsearch as part of OpenShift logging, the Jaeger Operator can use the installed Elasticsearch Operator to provision storage.

Table 5.4. Jaeger general storage parameters

Parameter	Description	Values	Default value

spec:
  storage:
    options: {}
	Configuration options that define the storage.		

storage: type:	Type of storage to use for the deployment.	**memory** or **elasticsearch**. Memory storage is only appropriate for development, testing, demonstrations, and proof of concept environments as the data does not persist if the pod is shut down. For production environments Jaeger supports Elasticsearch for persistent storage.	**memory**

Table 5.5. Elasticsearch configuration parameters

Parameter	Description	Values	Default value
General Elasticsearch configuration settings			
elasticsearch: server-urls:	URL of the Elasticsearch instance.	The fully-qualified domain name of the Elasticsearch server. If you have specified **spec:storage:type**=**elasticsearch** but have not specified a value for **server-urls** parameter, the Jaeger Operator will use the Elasticsearch Operator to create an Elasticsearch cluster using the configuration in the **spec:storage:elasticsearch** section of the custom resource file.	**http://elasticsearch.<namespace>.svc:9200**
es: max-num-spans:	The maximum number of spans to fetch at a time, per query, in Elasticsearch.		10000
es: max-span-age:	The maximum lookback for spans in Elasticsearch.		72h0m0s
elasticsearch: secretname:	Name of the secret, for example **jaeger-secret**.		N/A
es: sniffer:	The sniffer configuration for Elasticsearch. The client uses the sniffing process to find all nodes automatically. Disabled by default.	**true**/ **false**	**false**
es: timeout:	Timeout used for queries. When set to zero there is no timeout.		0s

Parameter	Description	Values	Default value
es: username:	The username required by Elasticsearch. The basic authentication also loads CA if it is specified. See also **es.password**.		
es: password:	The password required by Elasticsearch. See also, **es.username**.		
es: version:	The major Elasticsearch version. If not specified, the value will be auto-detected from Elasticsearch.		0
Elasticsearch resource configuration settings			
elasticsearch: nodeCount:	Number of Elasticsearch nodes. For high availability use at least 3 nodes. Do not use 2 nodes as "split brain" problem can happen.	Integer value. For example, Proof of concept = 1, Minimum deployment =3	1

elasticsearch:
  resources:
    requests:
      cpu:
	Number of central processing units for requests, based on your environment's configuration.	Specified in cores or millicores (for example, 200m, 0.5, 1). For example, Proof of concept = 500m, Minimum deployment =1	1Gi

elasticsearch:
  resources:
    requests:
      memory:
	Available memory for requests, based on your environment's configuration.	Specified in bytes (for example, 200Ki, 50Mi, 5Gi). For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*	500m

elasticsearch:
  resources:
    limits:
      cpu:
	Limit on number of central processing units, based on your environment's configuration.	Specified in cores or millicores (for example, 200m, 0.5, 1). For example, Proof of concept = 500m, Minimum deployment =1	

Parameter	Description	Values	Default value

elasticsearch:
  resources:
    limits:
      memory:
	Available memory limit based on your environment's configuration.	Specified in bytes (for example, 200Ki, 50Mi, 5Gi). For example, Proof of concept = 1Gi, Minimum deployment = 16Gi*	

	*Each Elasticsearch node can operate with a lower memory setting though this is NOT recommended for production deployments. For production use, you should have no less than 16Gi allocated to each Pod by default, but preferably allocate as much as you can, up to 64Gi per Pod.		
Elasticsearch data replication options			
elasticsearch: redundancyPolicy:	Data replication policy defines how Elasticsearch shards are replicated across data nodes in the cluster. If not specified,the Jaeger Operator automatically determines the most appropriate replication based on number of nodes.	**ZeroRedundancy**(no replica shards), **SingleRedundancy**(one replica shard), **MultipleRedundancy** (each index is spread over half of the Data nodes), **FullRedundancy** (each index is fully replicated on every Data node in the cluster).	
es: num-replicas:	The number of replicas per index in Elasticsearch.		1
es: num-shards:	The number of shards per index in Elasticsearch.		5
Elasticsearch index and index cleaner configuration options			
es: create-index- templates:	Automatically create index templates at application startup when set to **true**. When templates are installed manually, set to **false**.	**true**/ **false**	**true**

Parameter	Description	Values	Default value
es: index-prefix:	Optional prefix for Jaeger indices. For example, setting this to "production" creates indices named "production-jaeger-*".		
esIndexCleaner: enabled:	When using Elasticsearch storage, by default a job is created to clean old traces from the index. This parameter enables or disables the index cleaner job.	**true**/ **false**	**true**
esIndexCleaner: numberOfDays:	Number of days to wait before deleting an index.	Integer value	**7**
esIndexCleaner: schedule:	Defines the schedule for how often to clean the Elasticsearch index.	Cron expression	"55 23 * * *"
esRollover: schedule:	Defines the schedule for how often to rollover to a new Elasticsearch index.	Cron expression	'*/30 * * * *'
Configuration settings for Elasticsearch bulk processor			
es: bulk: actions:	The number of requests that can be added to the queue before the bulk processor decides to commit updates to disk.		1000
es: bulk: flush-interval:	A **time.Duration** after which bulk requests are committed, regardless of other thresholds. To disable the bulk processor flush interval, set this to zero.		200ms

Parameter	Description	Values	Default value

es:
  bulk:
    size:
	The number of bytes that the bulk requests can take up before the bulk processor decides to commit updates to disk.		5000000

es:
  bulk:
    workers:
	The number of workers that are able to receive and commit bulk requests to Elasticsearch.		1

Elasticsearch TLS configuration settings			

es:
  tls:
    ca:
	Path to a TLS Certification Authority (CA) file used to verify the remote server(s).		Will use the system truststore by default.

es:
  tls:
    cert:
	Path to a TLS Certificate file, used to identify this process to the remote server(s).		

es:
  tls:
    enabled:
	Enable transport layer security (TLS) when talking to the remote server(s). Disabled by default.	**true**/ **false**	**false**

es:
  tls:
    key:
	Path to a TLS Private Key file, used to identify this process to the remote server(s).		

es:
  tls:
    server-name:
	Override the expected TLS server name in the certificate of the remote server(s).		

es: token-file:	Path to a file containing the bearer token. This flag also loads the Certification Authority (CA) file if it is specified.		

Parameter	Description	Values	Default value
Elasticsearch archive configuration settings			

es-archive:
  bulk:
    actions:
	The number of requests that can be added to the queue before the bulk processor decides to commit updates to disk.		0

es-archive:
  bulk:
    flush-interval:
	A **time.Duration** after which bulk requests are committed, regardless of other thresholds. To disable the bulk processor flush interval, set this to zero.		0s

es-archive:
  bulk:
    size:
	The number of bytes that the bulk requests can take up before the bulk processor decides to commit updates to disk.		0

es-archive:
  bulk:
    workers:
	The number of workers that are able to receive and commit bulk requests to Elasticsearch.		0

es-archive:
  create-index-
  templates:
	Automatically create index templates at application startup when set to **true**. When templates are installed manually, set to **false**.	**true**/ **false**	**false**

es-archive: enabled:	Enable extra storage.	**true**/ **false**	**false**
es-archive: index-prefix:	Optional prefix for Jaeger indices. For example, setting this to "production" creates indices named "production-jaeger-*".		

Parameter	Description	Values	Default value
es-archive: max-num-spans:	The maximum number of spans to fetch at a time, per query, in Elasticsearch.		0
es-archive: max-span-age:	The maximum lookback for spans in Elasticsearch.		0s
es-archive: num-replicas:	The number of replicas per index in Elasticsearch.		0
es-archive: num-shards:	The number of shards per index in Elasticsearch.		0
es-archive: password:	The password required by Elasticsearch. See also, **es.username**.		
es-archive: server-urls:	The comma-separated list of Elasticsearch servers. Must be specified as fully qualified URLs, for example, **http://localhost:9200**.		
es-archive: sniffer:	The sniffer configuration for Elasticsearch. The client uses the sniffing process to find all nodes automatically. Disabled by default.	**true**/ **false**	**false**
es-archive: timeout:	Timeout used for queries. When set to zero there is no timeout.		0s
es-archive: tls: ca:	Path to a TLS Certification Authority (CA) file used to verify the remote server(s).		Will use the system truststore by default.

Parameter	Description	Values	Default value

es-archive:
  tls:
    cert:
	Path to a TLS Certificate file, used to identify this process to the remote server(s).		

es-archive:
  tls:
    enabled:
	Enable transport layer security (TLS) when talking to the remote server(s). Disabled by default.	**true**/ **false**	**false**

es-archive:
  tls:
    key:
	Path to a TLS Private Key file, used to identify this process to the remote server(s).		

es-archive:
  tls:
    server-name:
	Override the expected TLS server name in the certificate of the remote server(s).		

es-archive: token-file:	Path to a file containing the bearer token. This flag also loads the Certification Authority (CA) file if it is specified.		
es-archive: username:	The username required by Elasticsearch. The basic authentication also loads CA if it is specified. See also **es-archive.password**.		
es-archive: version:	The major Elasticsearch version. If not specified, the value will be auto-detected from Elasticsearch.		0

Production storage example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
```

```
    storage:
      type: elasticsearch
      elasticsearch:
        nodeCount: 3
        resources:
          requests:
            cpu: 1
            memory: 16Gi
          limits:
            memory: 16Gi
```

Storage example with volume mounts

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: https://quickstart-es-http.default.svc:9200
        index-prefix: my-prefix
        tls:
          ca: /es/certificates/ca.crt
    secretName: jaeger-secret
  volumeMounts:
    - name: certificates
      mountPath: /es/certificates/
      readOnly: true
  volumes:
    - name: certificates
      secret:
        secretName: quickstart-es-http-certs-public
```

Storage example with persistent storage:

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-prod
spec:
  strategy: production
  storage:
    type: elasticsearch
    elasticsearch:
      nodeCount: 1
      storage:                    ❶
        storageClassName: gp2
        size: 5Gi
      resources:
        requests:
          cpu: 200m
```

```
        memory: 4Gi
      limits:
        memory: 4Gi
    redundancyPolicy: ZeroRedundancy
```

[1] Persistent storage configuration. In this case AWS **gp2** with **5Gi** size. When no value is specified, Jaeger uses **emptyDir**. The Elasticsearch Operator provisions **PersistentVolumeClaim** and **PersistentVolume** which are not removed with Jaeger instance. You can mount the same volumes if you create a Jaeger instance with the same name and namespace.

5.1.1.5. Jaeger Query configuration options

Query is a service that retrieves traces from storage and hosts the user interface to display them.

Table 5.6. Jaeger Query parameters

Parameter	Description	Values	Default value

spec:
  query:
    options: {}
    resources: {}
	Configuration options that define the Query service.		

query: additional-headers:	Additional HTTP response headers. Can be specified multiple times.	Format: "Key: Value"	
query: base-path:	The base path for all jaeger-query HTTP routes can be set to a non-root value, for example, /**jaeger** would cause all UI URLs to start with /**jaeger**. This can be useful when running jaeger-query behind a reverse proxy.	/{path}	
query: port:	The port for the query service.		16686
options: log-level:	Logging level for Query.	Possible values: **trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**.	

Sample Query configuration

```
apiVersion: jaegertracing.io/v1
kind: "Jaeger"
metadata:
  name: "my-jaeger"
spec:
  strategy: allInOne
  allInOne:
    options:
      log-level: debug
      query:
        base-path: /jaeger
```

5.1.1.6. Jaeger Ingester configuration options

Ingester is a service that reads from a Kafka topic and writes to another storage backend (Elasticsearch). If you are using the **allInOne** or **production** deployment strategies, you do not need to configure the Ingester service.

Table 5.7. Jaeger Ingester parameters

Parameter	Description	Values

spec:
  strategy: streaming
  ingester:
    options: {}
	Configuration options that define the Ingester service.	

autoscale:	This parameter controls whether to enable/disable autoscaling for the Ingester. Autoscaling is enabled by default. Set to **false** to explicitly disable autoscaling.	**true**/**false**

kafka:
  consumer:
    topic:
	The **topic** parameter identifies the Kafka configuration used by the collector to produce the messages, and the ingester to consume the messages.	Label for the consumer. For example, **jaeger-spans**.

kafka:
  consumer:
    brokers:
	Identifies the Kafka configuration used by the Ingester to consume the messages.	Label for the broker, for example, **my-cluster-kafka-brokers.kafka:9092**.

Parameter	Description	Values
ingester: deadlockInterval:	Specifies the interval (in seconds or minutes) that the Ingester should wait for a message before terminating. The deadlock interval is disabled by default (set to 0), to avoid the Ingester being terminated when no messages arrive while the system is being initialized.	Minutes and seconds, for example, **1m0s**. Default value is **0**.
log-level:	Logging level for the Ingester.	Possible values: **trace**, **debug**, **info**, **warning**, **error**, **fatal**, **panic**.
maxReplicas:	Specifies the maximum number of replicas to create when autoscaling the Ingester.	Integer, for example, **100**.

Streaming Collector and Ingester example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  collector:
    options:
      kafka:
        producer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
  ingester:
    options:
      kafka:
        consumer:
          topic: jaeger-spans
          brokers: my-cluster-kafka-brokers.kafka:9092
      ingester:
        deadlockInterval: 5
  storage:
    type: elasticsearch
    options:
      es:
        server-urls: http://elasticsearch:9200
```

5.1.1.6.1. Configuring Ingester for autoscaling

You can configure the Ingester to autoscale; the Ingester will scale up or down based on the CPU and/or memory consumption. Configuring the Ingester to autoscale can help you ensure your Jaeger environment scales up during times of increased load, and scales down when less resources are needed, saving on costs. You configure autoscaling by setting the **autoscale** parameter to **true** and specifying a value for **.spec.ingester.maxReplicas** along with a reasonable value for the resources that you expect the Ingester's pod to consume. If you do not set a value for **.spec.ingester.maxReplicas** the Operator will set it to **100**.

By default, when there is no value provided for **.spec.ingester.replicas**, the Jaeger Operator creates a Horizontal Pod Autoscaler (HPA) configuration for the Ingester. For more information about HPA, refer to the Kubernetes documentation.

The following is an example autoscaling configuration, setting the Ingester's limits as well as the maximum number of replicas:

Ingester autoscaling example

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: simple-streaming
spec:
  strategy: streaming
  ingester:
    maxReplicas: 8
    resources:
      limits:
        cpu: 100m
        memory: 128Mi
```

5.1.2. Injecting sidecars

OpenShift Jaeger relies on a proxy sidecar within the application's pod to provide the agent. The Jaeger Operator can inject Jaeger Agent sidecars into Deployment workloads. You can enable automatic sidecar injection or manage it manually.

5.2. AUTOMATICALLY INJECTING SIDECARS

To enable this feature, you add the annotation **sidecar.jaegertracing.io/inject** set to either the string **true** or the Jaeger instance name as returned by **oc get jaegers**. When you specify **true**, there should be only a single Jaeger instance for the same namespace as the deployment, otherwise, the Operator cannot determine which Jaeger instance to use. A specific Jaeger instance name on a deployment has a higher precedence than **true** applied on its namespace.

The following snippet shows a simple application that will inject a sidecar, with the Jaeger Agent pointing to the single Jaeger instance available in the same namespace:

sample automatic sidecar injection

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
```

```
      "sidecar.jaegertracing.io/inject": "true"  1
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: acme/myapp:myversion
```

When the sidecar is injected, the Jaeger Agent can then be accessed at its default location on **localhost**.

5.3. MANUALLY INJECTING SIDECARS

For controller types other than **Deployments** (for example, **StatefulSets**, **DaemonSets**, etc.), you can manually define the Jaeger Agent sidecar in your specification.

The following snippet shows the manual definition you can include in your containers section for a Jaeger Agent sidecar:

example sidecar definition for a StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: example-statefulset
  namespace: example-ns
  labels:
    app: example-app
spec:

  spec:
    containers:
      - name: example-app
        image: acme/myapp:myversion
        ports:
          - containerPort: 8080
            protocol: TCP
      - name: jaeger-agent
        image: registry.redhat.io/distributed-tracing/jaeger-agent-rhel7:<version>
          # The agent version must match the operator version
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 5775
            name: zk-compact-trft
            protocol: UDP
          - containerPort: 5778
            name: config-rest
            protocol: TCP
          - containerPort: 6831
```

```
        name: jg-compact-trft
        protocol: UDP
      - containerPort: 6832
        name: jg-binary-trft
        protocol: UDP
      - containerPort: 14271
        name: admin-http
        protocol: TCP
    args:
      - --reporter.grpc.host-port=dns:///jaeger-collector-headless.example-ns:14250
      - --reporter.type=grpc
```

The Jaeger Agent can then be accessed at its default location on localhost.

5.4. UPGRADING JAEGER

The Operator Lifecycle Manager (OLM) controls the installation, upgrade, and role-based access control (RBAC) of Operators in a cluster. The OLM runs by default in OpenShift Container Platform. The OLM queries for available Operators as well as upgrades for installed Operators. For more information about how OpenShift Container Platform handled upgrades, refer to the Operator Lifecycle Manager documentation.

The update approach used by the Jaeger Operator upgrades the managed Jaeger instances to the version associated with the Operator. Whenever a new version of the Jaeger Operator is installed, all the Jaeger application instances managed by the Operator will be upgraded to the Operator's version. For example, if version 1.10 is installed (both Operator and backend components) and the Operator is upgraded to version 1.11, then as soon as the Operator upgrade has completed, the Operator will scan for running Jaeger instances and upgrade them to 1.11 as well.

5.5. REMOVING JAEGER

The steps for removing Jaeger from an OpenShift Container Platform cluster are as follows:

1. Shut down any Jaeger pods.

2. Remove any Jaeger instances.

3. Remove the Jaeger Operator.

5.5.1. Removing a Jaeger instance using the web console

> **NOTE**
>
> When deleting an instance that uses the in-memory storage, all data will be permanently lost. Data stored in a persistent storage (such as Elasticsearch) will not be deleted when a Jaeger instance is removed.

Procedure

1. Log in to the OpenShift Container Platform web console.

2. Navigate to **Operators → Installed Operators**.

3. Select the name of the project where the Operators are installed from the Project menu, for example, **jaeger-system**.

4. Click the Jaeger Operator.

5. Click the **Jaeger** tab.

6. Click the Options menu ⋮ next to the instance you want to delete and select **Delete Jaeger**.

7. In the confirmation message, click **Delete**.

5.5.2. Removing a Jaeger instance from the CLI

1. Log in to the OpenShift Container Platform CLI.

   ```
   $ oc login
   ```

2. To display the Jaeger instances run the command:

   ```
   $ oc get deployments -n <jaeger-project>
   ```

   The names of operators have the suffix **-operator**. The following example shows two Jaeger Operators and four Jaeger instances:

   ```
   $ oc get deployments -n jaeger-system
   ```

   You should see output similar to the following:

   ```
   NAME                   READY  UP-TO-DATE  AVAILABLE  AGE
   elasticsearch-operator  1/1    1           1          93m
   jaeger-operator         1/1    1           1          49m
   jaeger-test             1/1    1           1          7m23s
   jaeger-test2            1/1    1           1          6m48s
   tracing1                1/1    1           1          7m8s
   tracing2                1/1    1           1          35m
   ```

3. To remove an instance of Jaeger, run the command:

   ```
   $ oc delete jaeger <deployment-name> -n <jaeger-project>
   ```

   For example,

   ```
   $ oc delete jaeger tracing2 -n jaeger-system
   ```

4. To verify the deletion, run **oc get deployment** again:

   ```
   $ oc get deployments -n <jaeger-project>
   ```

   For example,

```
$ oc get deployments -n jaeger-system
```

Should generate output similar to the following:

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch-operator   1/1     1            1           94m
jaeger-operator          1/1     1            1           50m
jaeger-test              1/1     1            1           8m14s
jaeger-test2             1/1     1            1           7m39s
tracing1                 1/1     1            1           7m59s
```

5.5.3. Removing the Jaeger Operator

Procedure

1. Follow the instructions for Deleting Operators from a cluster .

   - Remove the Jaeger Operator.

   - After the Jaeger Operator has been removed, if appropriate, remove the Elasticsearch Operator.